Low-Complexity Codes for Random and Clustered High-Order Failures in Storage Arrays

Yuval Cassuto, *Member, IEEE* and Jehoshua Bruck, *Fellow, IEEE*

Abstract—RC (Random/Clustered) codes are a new efficient array-code family for recovering from 4-erasures. RC codes correct most 4-erasures, and essentially all 4-erasures that are clustered. Clustered erasures are introduced as a new erasure model for storage arrays. This model draws its motivation from correlated device failures, that are caused by physical proximity of devices, or by age proximity of endurance-limited solid-state drives. The reliability of storage arrays that employ RC codes is analyzed and compared to known codes. The new RC code is significantly more efficient, in all practical implementation factors, than the best known 4-erasure correcting MDS code. These factors include: small-write update-complexity, full-device update-complexity, decoding complexity and number of supported devices in the array.

Index Terms—Array codes, clustered erasures, correlated failures, storage arrays

1 Introduction

PROTECTING disk arrays and other dynamic storage systems against device failures has long become an essential part of their design. Implemented solutions to data availability in the presence of failed hardware have progressed considerably in the last two decades. The main technique to protect data against device failures is the application of erasure-correcting codes, that use redundant storage to protect against certain device failures. The initial market offerings of redundant disk arrays employed the *single-parity code* on one end of the spectrum, and the *repetition code* on the other end. The single parity code requires the inclusion of one redundant device per array, and protects against any single device failure. The repetition code mirrors each device in a redundant device, and protects against all failures in the original devices. These two schemes are referred to in the storage literature as RAID-5 and RAID-1, respectively[1]. As it turns out, RAID-5 provides insufficient failure protection, and RAID-1 is too costly in the amount of required redundant storage. *Codes for double erasures* were thus the next choice, and are implemented in RAID-6 products. As the capacity of storage devices grows, it is likely that individual-device reliability will degrade (note that when the device capacity grows, the Sector Error Rate (SER) needs to decrease in order to maintain the same device reliability). Moreover, even maintaining the same SER for higher areal densities is proving challenging, as heads, media, and servo struggle to scale with ambitious track and bit densities in hard-disk drives. Counting on the same reliability becomes even riskier when new storage technologies are introduced, such as solid-state drives, as well as emerging novel magnetic-disk recording technologies such as *Patterned Media* [7] and *Heat Assisted Recording* [11]. Solid state drives, in particular, equipped with an indirection system to control device wear, are exposed to a new set of potential reliability impediments such as meta-data loss, data mislocation and retention failures. Consequently, it is likely that, at least for some applications, a stronger failure protection will be required at the system level.

All of the codes considered for storage arrays to this date, assume that device failures are independent, and are therefore designed for correction of a given number of failures, with no respect to *which* devices fail. However, with the increase of the failure protection requirement from the codes, we also need to examine the failure models that these codes are designed to tackle. A justification of the need to consider new failure models is provided by the following two facts.

1) Assuming independent failures accounts for the stationary part of the failure process, but excludes ”catastrophic” failure events.
2) Known codes for high failure correction have significant implementation complexity, in all relevant complexity measures: encoding, decoding, and updates.

These two facts imply that moving to high failure correction without aiming at more realistic failure models is sub-optimal, since it enforces a stronger than needed failure model, that comes at a very high implementation cost. For example, correcting 4 simultaneous *independent* failures may be an

- *Y. Cassuto is with Hitachi Global Storage Technologies, San Jose Research Center, San Jose, CA, 95135.*
  E-mail: yuval.cassuto@hitachigst.com
- *J. Bruck is with the California Institute of Technology, Department of Electrical Engineering, Pasadena, CA, 91125.*
  E-mail: bruck@paradise.caltech.edu

This work was supported in part by the Caltech Lee Center for Advanced Networking

1. The acronym RAID stands for Redundant Array of Inexpensive Disks [14]

excessive design goal, while the ability to correct 4 failures due to a rare catastrophic event may actually be a desired property. For such catastrophic events, the assumption that device failures occur independently of each other is no longer true, and many failure mechanisms cause multiple device failures that are highly correlated. Since adjacent devices in the array share cabling, cooling, power and mechanical stress, failure combinations that are clustered are more likely than completely isolated multiple failures. Another motivation to depart from the independent-failure model are solid-state drives, whose limited endurance makes devices of similar age fail at similar times. Consequently, codes that have excellent clustered-failure correctability, good independent (a.k.a random)-failure correctability are sought, and proposed by the main code construction of the paper.

The new erasure-correcting code family, introduced in this paper, is from the class of *array codes*. The class of codes called array codes [4] are the best fit for storage applications, since they enjoy both simple decoding and efficient updates, while requiring low storage redundancy. For an introductory example of array codes, please refer to Appendix A. In the literature of array codes a column serves as an abstraction to a storage device or another physical storage unit, and column erasures represent device failures. The current state-of-the-art in storage array codes, that will be used herein as comparison to the new codes, are the EVENODD codes [2], that protect p information columns against two column erasures, for any prime number p. The EVENODD family of codes and its relatives (e.g. [6]), can recover from any two erasures with optimal redundancy (MDS), and enjoy simple decoding and fairly low, though not optimal, update complexity. EVENODD codes for more than two erasures exist [3], but their decoding becomes more complex for growing numbers of erasures, and their update complexity grows as fast as $2r-1$, for r correctable erasures. A high update complexity limits the system performance as it imposes excess disk I/O operations, even if no failures occur. High update complexity also implies high wear of parity devices whose direct consequence is the shortening of device lifetimes.

To obtain a precise model definition for correlated failures, in sub-section 2.2 a new classification of erasure combinations is proposed. Each combination of column erasures will be classified by the number of erased columns, and by the number of *clusters* in which the erased columns fall. The number of clusters captures the number of "independent" failure events, each possibly affecting multiple devices in a single cluster.

The main contribution of this paper, detailed in section 4, is the construction of an array-code family, called RC codes (for Random/Clustered), that corrects essentially all clustered failures, and 7 out of 8 random, 4-failure combinations. The correction properties are stated and proved in section 5. Efficient decoding of RC codes is described in section 7. In section 6 RC-coded device arrays are compared to EVENODD-coded ones in terms of their expected reliability, by analyzing their respective Mean Time To Data Loss ($MTTDL$), under random and clustered failures. A detailed comparison of the properties of RC codes and EVENODD($r=4$) codes is provided in section 8. In summary, the RC codes are better than EVENODD($r=4$) in all implementation complexity parameters. They improve the encoding and decoding complexities by 25%, and the small-write update complexity by 28.57%. They also support twice as many devices compared to EVENODD codes, for the same column size.

2 Definitions and Notations

2.1 Array codes

The definitions in this sub-section are standard coding-theory terminology that provides a good abstraction for failure-resilient storage arrays. A *length* n array code consists of n columns. A column is a model for, depending on the exact application, either a whole device or a different physical data unit in the storage array. In the codes discussed here, there are k columns that store uncoded information bits and r columns that store redundant parity bits (thus $n=k+r$). This array structure has the advantage that information can be read off a device directly without decoding, unless it suffered a failure, in which case a decoding process is invoked. An array code that admits this structure is called *strongly systematic*. A *column erasure* occurs when, for some physical reason, the contents of a particular column cannot be used by the decoder. An erasure is a model for a device failure whereby all the data on the device (or other physical unit) is known to have become unusable. We say that an array with given column erasures is *correctable* by the array code if there exists a decoding algorithm that, independent of the specific array contents, can reconstruct the original array from unerased columns only. An array code is called MDS (*Maximum Distance Separable*) if it has r redundant columns and it can correct all possible combinations of r column erasures. MDS codes obviously have the strongest conceivable erasure correction capability for a given redundancy, since the k information columns can be recovered from *any* k columns. Beyond space efficiency of the code, one should also consider its I/O efficiency. I/O efficiency of a disk array is determined by the *small-write* and *full-column* update complexities of the array code used. The small-write update complexity (often simply called update complexity) is defined as the number of parity-bit updates required for a single information bit update, averaged over all information bits. Appendix A shows how the small-write update complexity is calculated for a sample array code. The full-column update complexity is the number of parity columns that have to be modified per a single full-column update. Another crucial performance measure of an array code is its *erasure-decoding complexity*, defined as the number of bit operations (additions, shifts) required to recover the erased columns from the surviving ones. Unless noted otherwise, p will refer to a general prime number p.

2.2 Random/Clustered erasure correction

To describe column erasure combinations whose only restriction is the number of erased columns, it is customary to use the somewhat misleading term *random* [13] erasures.

Definition 1 *An array is said to recover from ρ* ***random erasures*** *if it can correct all combinations of ρ erased columns.*

The random erasure model is most natural when storage nodes are known to, or more commonly, assumed to behave uniformly and independent of each other. Indeed, almost all array-code constructions discussed in the literature aim at correcting random erasures. Refinement of the erasure model is possible by adding restrictions on the relative locations of the erased columns. This paper considers *clustered* erasures, where the ρ erasures fall into a limited ($< \rho$) number of clusters. We now turn to some definitions related to the clustered erasure model. In words, a *cluster* is a contiguous block of columns. More precisely,

Definition 2 *In an array code with columns numbered* $\{0, 1, 2, \ldots, n-1\}$, *a* ***cluster*** *is a set of* σ *columns such that the difference between the highest numbered column and the lowest numbered one is exactly* $\sigma - 1$.

For example, $\{2, 3, 4, 5\}$ is a cluster with $\sigma = 4$. Now given a set of columns, the number of clusters that it occupies is the partition of that set to a minimal number of subsets, each of which is a cluster according to the definition above. Now we include another definition that will be useful later.

Definition 3 *A set of* ρ *columns is called* ***clustered*** *if the number of clusters it occupies is* strictly *less than* ρ.

Random erasures have no restriction on their respective numbers of clusters and therefore they include both clustered and non-clustered erasures. The other extreme is the *column burst* model, where all erased columns need to fall into a single cluster. These two well-studied extreme cases open our presentation, and later the RC codes are shown to be very effective for all intermediate cases of clustered erasures. An illustration of the column-clustering definitions is given in Figure 1.

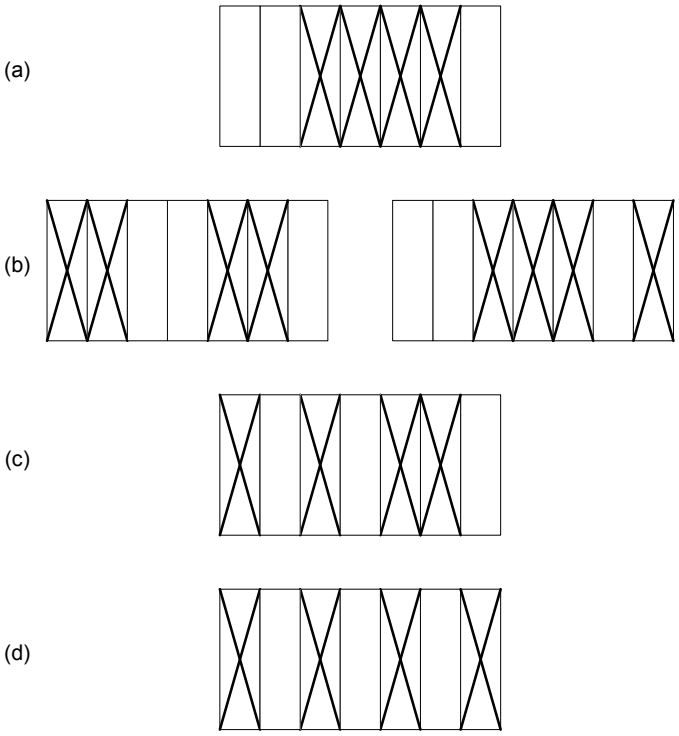

Fig. 1. Classification of column combinations by their respective numbers of clusters. Four columns (marked with X) that fall into (a) One cluster (b) Two clusters (c) Three clusters (d) Four clusters (non-clustered)

3 Preliminaries and Relevant Known Results

The purpose of this section is to discuss relevant known results in sufficient detail to prepare for the presentation of the new RC codes in the next section. Some algebraic tools used later for RC codes have been used before for other codes, so this section also serves to elucidate those tools prior to their use by the actual code construction.

3.1 Codes for erasures in a single cluster

Assume that our design goal is a storage array that will sustain any erasure of ρ columns in a single cluster, without requiring any random erasure correction capability. One option is to take a code that corrects any ρ *random* column erasures that, in particular, corrects any clustered ρ erasures. However, as can be expected, this may not be the best approach since correcting all random erasures is a far stronger requirement that excludes much simpler and more efficient constructs. As this section shows, a simple and well known technique called *interleaving* can achieve that task optimally both with respect to the required redundancy and in terms of the code update complexity.

Let $\mathcal{CP}$ be an array code with n' columns, out of which $k' = n' - 1$ are information columns. The remaining column holds the bit-wise parity of the k' information columns. Define the code $\mathcal{CP}_\rho$ as the length $n = \rho n'$ code that is obtained by the interleaving of ρ codewords of $\mathcal{CP}$. In other words, if $\mathsf{C}^{(1)}, \mathsf{C}^{(2)}, \ldots, \mathsf{C}^{(\rho)}$ are ρ codewords of $\mathcal{CP}$, then the corresponding code word of $\mathcal{CP}_\rho$ will be

$\mathsf{C}_1^{(1)}$	$\mathsf{C}_1^{(2)}$	$\cdots$	$\mathsf{C}_1^{(\rho)}$	$\mathsf{C}_2^{(1)}$	$\mathsf{C}_2^{(2)}$	$\cdots$	$\mathsf{C}_2^{(\rho)}$	$\mathsf{C}_3^{(1)}$	$\cdots$

Proposition 1 *The code* $\mathcal{CP}_\rho$ *corrects any* ρ *erasures in a single cluster.*

Proof: Any erasure that is confined to at most ρ consecutive columns erases at most one column of each constituent $\mathcal{CP}$ code. These single erasures are correctable by the individual $\mathcal{CP}$ codes. □

It is clear that the code $\mathcal{CP}_\rho$ has optimal redundancy since it satisfies $\rho = r$ and ρ is a well known and obvious lower bound on the redundancy r. For any ρ, the code $\mathcal{CP}_\rho$ has update complexity (both small-write and full-column) of 1, which is optimal since a lower update complexity would imply at least one code bit that is independent of all other bits, and erasure of that bit would not be correctable.

3.2 Codes for random erasures: EVENODD

As mentioned in sub-section 3.1, array codes that correct any ρ random erasures also correct any ρ clustered erasures. In this section we seek to survey a family of random erasure

correcting codes: the EVENODD [2],[3] codes. These codes enjoy several properties that make them most appealing for implementation in storage arrays. The purpose of presenting the codes here is twofold. First, in the absence of prior codes that differentiate clustered and random correction capabilities, EVENODD will be used as the current state-of-the-art for comparison with our construction. Second, the analysis of the new RC codes herein is made simpler by building on the algebraic framework previously developed for EVENODD. An EVENODD code [2] takes p data columns, each of size $p-1$ and adds to them 2 parity columns of the same size. The encoded array is therefore of size $(p-1)\times(p+2)$. EVENODD can correct any 2 column erasures so it is clearly optimal in terms of added redundancy (MDS). Other properties of EVENODD are that it is strongly systematic, it has relatively low (but not optimal) small-write update-complexity, and optimal full-column update complexity. In addition, it can be encoded and decoded using simple XOR and shift operations. For complete geometric and algebraic definitions of EVENODD codes, please consult Appendix B. EVENODD codes have been generalized to high-order ($r \geq 3$) random erasure correction in [3]. The main idea in the generalization is to add more parity columns that constrain the code bits across diagonals with different slopes (recall that the base EVENODD uses slopes 0 and 1). Discussing EVENODD generalization in depth is beyond the scope of this paper. We only mention the following facts that are relevant to our presentation.

- The asymptotic small-write update-complexity of the general EVENODD code family is $2r-1-o(1)$. $o(1)$ refers to terms that tend to zero as the code length goes to infinity. Their full-column update-complexity is r.
- For $r > 3$, generalized r random erasure correcting EVENODD codes are only guaranteed to exist for p that belong to a subset of the primes: those that satisfy that 2 is a primitive element in the Galois field GF(p).
- The best known way to decode general EVENODD codes is using the algorithm of [5] over the ring $\mathcal{R}_p$, for which the decoding complexity is dominated by the term rkp.

3.3 Mathematical framework

We now describe the mathematical framework, borrowed from [5], to present the new RC codes. The length $p-1$ columns of the code array are viewed as polynomials of degree $\leq p-2$ over the finite field $\mathbb{F}_2$, taken modulo the polynomial $M_p(x)$, where $M_p(x) = (x^p+1)/(x+1) = x^{p-1}+x^{p-2}+\cdots+x+1$ (recall that in $\mathbb{F}_2$ summation and subtraction are the same and both done using the boolean XOR function). According to that view, the polynomial for a binary column vector $\boldsymbol{c} = [c_0,\ldots,c_{p-2}]^T$ is denoted $c(\alpha) = c_{p-2}\alpha^{p-2}+\cdots+c_1\alpha+c_0$. Bit-wise addition modulo 2 of two columns is equivalent to summing the corresponding polynomials in the ring of polynomials modulo $M_p(x)$, denoted $\mathcal{R}_p$. Multiplying $c(\alpha)$ by α results in a downward shift of $\boldsymbol{c}$ if c_{p-2} is zero. In the case $c_{p-2}=1$, multiplying by α requires reduction modulo $M_p(x)$ and thus $\alpha c(\alpha)$ is obtained by first downward shifting $[c_0,\ldots,c_{p-3},0]^T$, and then inverting all the bits of the shifted vector. The RC code's encoding rules comprise column bit-wise additions and column shift-and-invert, hence representing these exact operations as addition and multiplication, respectively, over the ring $\mathcal{R}_p$ endows the codes with useful algebraic structure to analyze its properties.

Throughout the paper, we will assume that the prime number p is chosen such that 2 is irreducible in GF(p), hence $M_p(x)$ is irreducible, and the ring $\mathcal{R}_p$ becomes a finite field [12]. This assumption will simplify the correctability proofs in section 5. The correctability of erasure patterns under RC codes are proved by showing that the determinant of a submatrix of the code parity-check matrix has an inverse over the ring (field) $\mathcal{R}_p$.

4 Definition of the RC Codes

4.1 Geometric description

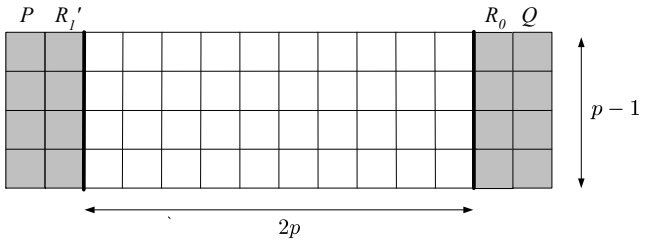

Fig. 2. The RC-code array. RC codes have $2p$ information columns and 4 parity columns. The column size is $p-1$.

Referring to Figure 2, the RC code has $2p$ information columns (white) of $p-1$ bits each and 4 parity columns (shaded) with the same number of bits. The information columns are numbered in ascending order from left to right using the integers $\{0,1,2,\ldots,2p-1\}$. Parity columns are not numbered and we use letter labels for them: $\{P, R_1', R_0, Q\}$. The code is defined using its encoding rules shown in Figure 3, for the case $p=5$. An imaginary row is added to the array to show the structure of the encoding function. Each shape represents, similarly to the definition of EVENODD in Appendix B, a group of bits that are constrained by the code to have even/odd parities. In other words, each parity bit is calculated from all the information bits that carry the same shape. Parity column P, located in the left most column of the left parity section, is simply the bit-wise even parity of the $2p$ information columns. Parity column R_1', located second from left, is the slope -1 diagonal parity of the *odd* numbered information columns $\{1,3,\ldots,2p-1\}$. The bit groups of R_1' are set to have even parity if the bits marked EO have even parity, and odd parity otherwise. Parity column R_0, located in the left most column of the right parity section, is the slope 2 diagonal parity of the *even* numbered information columns $\{0,2,\ldots,2p-2\}$. Parity column Q, located in the right most column of the right parity section, is the XOR of the slope 1 diagonal parities of both the even numbered columns and the odd numbered columns. The parity groups of Q and R_0, similarly to those of R_1', are set to be even/odd, based on the parity of the corresponding EO groups. Note that parity columns P and Q can be decomposed into $P = P0 \oplus P1$ and

$Q = Q0 \oplus Q1$, respectively, where $P0, Q0$ depend only on even information columns and $P1, Q1$ only on odd ones.

For a formal definition of the code we write the encoding functions explicitly. Denote by $c_{i,j}$ the bit in location i in information column j. For an integer l, define $\langle l \rangle$ to be $l \pmod p$. Now we write the explicit expression of the parity bits.

$$P_i = \bigoplus_{j=0}^{2p-1} c_{i,j}$$

$$R'_{1_i} = S_1 \oplus \bigoplus_{j=0}^{p-1} c_{\langle i+j \rangle, 2j+1} \ ,$$

$$\text{where} \qquad S_1 = \bigoplus_{j=0}^{p-1} c_{\langle p-1+j \rangle, 2j+1}$$

$$R_{0_i} = S_0 \oplus \bigoplus_{j=0}^{p-1} c_{\langle i-2j \rangle, 2j} \ ,$$

$$\text{where} \qquad S_0 = \bigoplus_{j=0}^{p-1} c_{\langle p-1-2j \rangle, 2j}$$

$$Q_i = S_Q \oplus (\bigoplus_{j=0}^{p-1} c_{\langle i-j \rangle, 2j}) \oplus (\bigoplus_{j=0}^{p-1} c_{\langle i-j \rangle, 2j+1}) \ ,$$

$$\text{where} \qquad S_Q = (\bigoplus_{j=0}^{p-1} c_{\langle p-1-j \rangle, 2j}) \oplus (\bigoplus_{j=0}^{p-1} c_{\langle p-1-j \rangle, 2j+1})$$

The encoding of information bits into an RC code array is illustrated in the example of Figure 4.

4.2 Algebraic description

Using the ring $\mathcal{R}_p$, the parity check matrix H of the RC code for $p = 5$ is given by

$$\left[\begin{array}{cc|cccccccccc|cc} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & \alpha^4 & 0 & \alpha^3 & 0 & \alpha^2 & 0 & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 & \alpha^2 & 0 & \alpha^4 & 0 & \alpha & 0 & \alpha^3 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 & \alpha^3 & \alpha^3 & \alpha^4 & \alpha^4 & 0 & 1 \end{array}\right]$$

The correspondence between the encoding function in Figure 3 and the parity check matrix above is straightforward. The columns of the parity check matrix correspond to columns of the code array. The two left most columns are for parity columns P and R'_1 and the two right most columns are for R_0 and Q. Columns in between correspond to information columns in the array. In the parity check matrix, row 1 represents the constraints enforced by parity column P, rows $2, 3, 4$ similarly represent the parity constraints of R'_1, R_0, Q, respectively. In any row j, the difference of exponents of α in two different columns is exactly the relative vertical shift of the two columns in the shape layout of the appropriate parity in Figure 3. For example, in the top row, all information columns have the same element, $1 (= \alpha^0)$, to account for the identical vertical alignment of the shapes in the encoding rule of parity

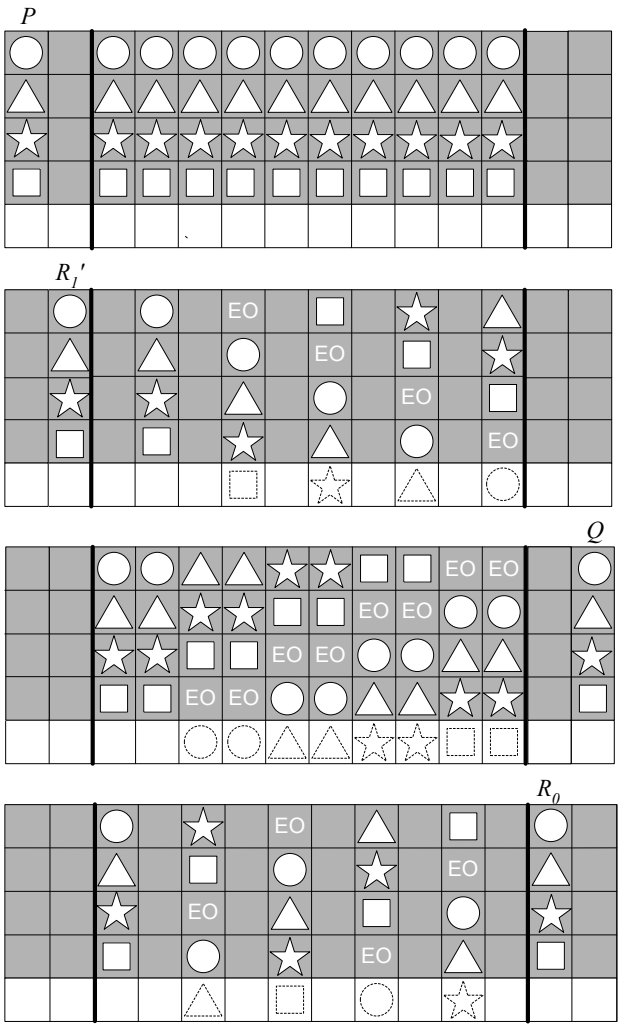

Fig. 3. Encoding of the RC code. From top to bottom: the parity groups of parity columns P (slope 0), R'_1 (slope -1), Q (slope 1) and R_0 (slope 2). Parity columns R_0 and R'_1 each depends on only half of the columns, contributing to the low implementation complexity of RC codes.

P. For general p the parity check matrix H has the following form.

$$H = \left[\begin{array}{cc|ccccccccc|cc} 1 & 0 & 1 & 1 & \cdots & 1 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & \alpha^{-i} & 0 & \alpha^{-(i+1)} & \cdots & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & \alpha^{2i} & 0 & \alpha^{2(i+1)} & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & \cdots & \alpha^{i} & \alpha^{i} & \alpha^{i+1} & \alpha^{i+1} & \cdots & \alpha^{p-1} & 0 & 1 \end{array}\right] \tag{1}$$

After presenting the RC code family, we proceed in the next section to prove its random and clustered erasure correction capabilities.

5 Erasure Correctability of RC Codes

In this section we prove that essentially all clustered combinations of 4 erasures are correctable by RC codes. Moreover, considering random erasure correctability, we prove that a $7/8$ portion of *all* combinations of 4 erasures are correctable by RC codes.

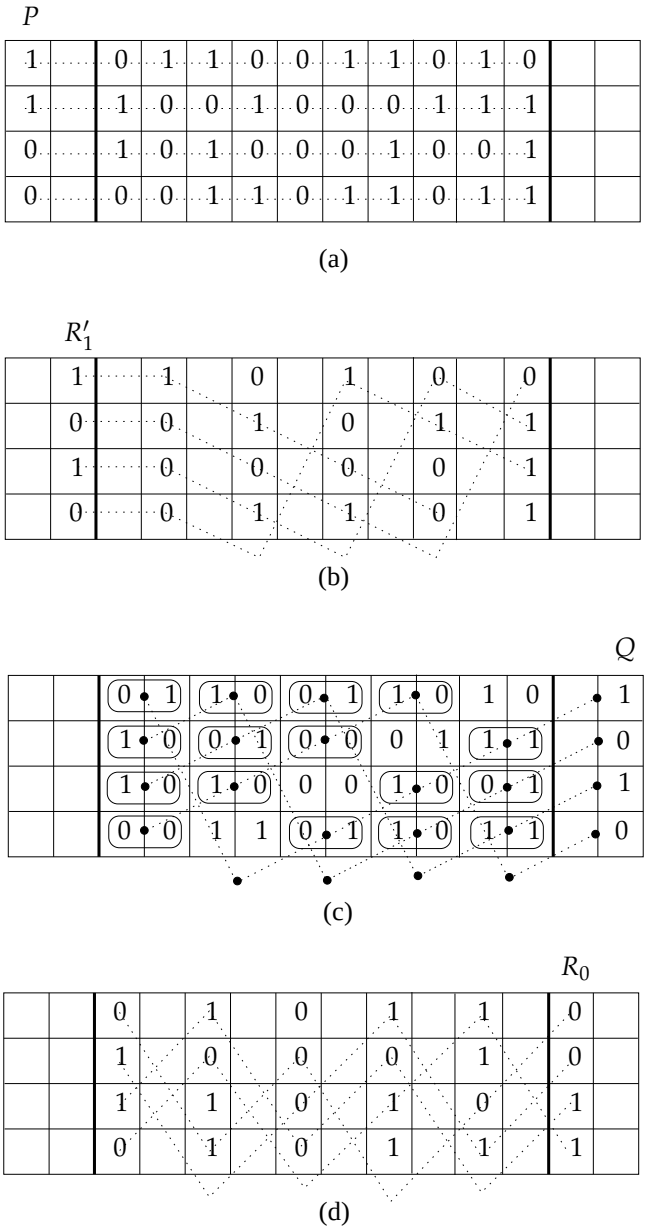

Fig. 4. Encoding example. Each parity group from Figure 3 is shown here traversed by a dotted line. (a) Parity column P (always even parity) (b) The groups of parity column R'_1 have odd parity since the non-traversed bit group has an odd number of ones. (c) The groups of parity column Q have even parity since the non-traversed bit group has an even number of ones. (d) The groups of parity column R_0 have odd parity since the non-traversed bit group has an odd number of ones.

5.1 Clustered erasure correctability

We first prove RC codes' excellent correction capability of clustered erasures. This result is established in Theorems 5 and 6 below that follow a sequence of lemmas. Recall that the $2p+4$ columns of the RC codes are labeled $\{P, R'_1, 0, 1, \ldots, 2p-2, 2p-1, R_0, Q\}$.

Lemma 2 *For a combination of* 4 *erasures, if* 3 *columns are either even numbered information columns or parity columns in* $\{R_0, P, Q\}$*, and* 1 *column is an odd numbered information column or the parity column* R'_1*, then it is a correctable* 4*-erasure. The complement case,* 3 *odd (or* R'_1 *or* P *or* Q*) and* 1 *even (or* R_0*), is correctable as well. (in particular, any 3-erasure is correctable).*

Proof: The RC code can correct the erasure patterns under consideration using a two-step procedure. The first step is to recover the erased odd information column. Since only one odd column is erased, parity column R'_1 can be used to easily recover all of its bits. Then, when all odd columns are available, $P1$ and $Q1$ are computed, and used to find $P0$ and $Q0$ from P and Q (if not erased) by

$$P0 = P1 \oplus P \quad , \quad Q0 = Q1 \oplus Q$$

After that step, between even information columns, R_0, $P0$ and $Q0$, only 3 columns are missing. Since even columns, R_0, $P0$ and $Q0$ constitute an EVENODD code with $r = 3$, the 3 missing columns can be recovered. The complement case of 3 odd and 1 even column erasures is settled by observing that odd columns, R'_1, $P1$ and $Q1$ constitute an $r = 3$ MDS code [10]. □

The next Lemma presents the key property that gives RC codes their favorable random and clustered erasure correctability.

Lemma 3 *When* $p > 5$*, for a combination of* 4 *erasures, if* 2 *columns are even numbered information columns and* 2 *columns are odd numbered information columns, then it is a correctable* 4*-erasure.*

Proof: For the case of 2 even and 2 odd information column erasures we consider two erasure patterns. All possible erasure combinations of that kind are either covered by these patterns directly or are equivalent to them in a way discussed below. The discussion of each erasure pattern will commence with its representing diagram. In these diagrams, a column marked 0 represents an even column and a column marked 1 represents an odd column. Between each pair of columns, an expression for the number of columns that separate them is given .

a) Erasures of the form

0	$\leftarrow 2j \rightarrow$	1	$\leftarrow 2(k-1) \rightarrow$	0	$\leftarrow 2l \rightarrow$	1	

The variables j, k, l satisfy the following conditions: $1 \leq k$, $1 \leq j+k+l \leq p-1$.

The location of the first even erased column, together with j, k, l determine the locations of the 4 erasures. Any even cyclic shift of the diagram above does not change the correctability of the erasure pattern since this shift gives the same sub-matrix of the parity-check matrix, up to a non-zero multiplicative constant. Hence, we can assume, without loss of generality, that the first even erased column is located in the leftmost information column. To prove the correctability of this erasure pattern we examine the determinant (over $\mathcal{R}_p$)

of the square sub-matrix of H, that corresponds to the erased columns. This determinant is itself an element of $\mathcal{R}_p$ and if it is provably *invertible* in $\mathcal{R}_p$, then the erasure combination is correctable by the RC code.

The sub-matrix that corresponds to the erasure pattern above is

$$\mathcal{M}_a^{(j,k,l)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha^{-j} & 0 & \alpha^{-j-k-l} \\ 1 & 0 & \alpha^{2(j+k)} & 0 \\ 1 & \alpha^{j} & \alpha^{j+k} & \alpha^{j+k+l} \end{bmatrix}.$$

Evaluating the determinant of this matrix gives

$$\begin{aligned} \left|\mathcal{M}_a^{(j,k,l)}\right| &= \alpha^{2j+3k+l} + \alpha^{2j+k-l} + \alpha^{j+2k} + \alpha^{j+k-l} + \\ &\quad + \alpha^{k+l} + \alpha^{k} + \alpha^{-l} + \alpha^{-k-l} \\ &= \alpha^{-l}(\alpha^{j+k}+1)(\alpha^{k+l}+1)\cdot \\ &\quad \cdot(\alpha^{j+k+l} + \alpha^{j} + \alpha^{l} + 1 + \alpha^{-k}) \end{aligned}$$

Given the ranges of the variables j,k,l, neither of the terms in the product above can evaluate to zero. Hence the determinant is invertible and this pattern is correctable.

b) Erasures of the form

0	$\leftarrow 2j-1 \rightarrow$	0	$\leftarrow 2k \rightarrow$	1	$\leftarrow 2l-1 \rightarrow$	1	

The variables j,k,l satisfy the following conditions: $1 \leq j$, $1 \leq l$, $1 \leq j+k+l \leq p-1$.

Here, like in the previous pattern, we assume, without loss of generality, that the first even erased column is located in the leftmost information column.

The sub-matrix that corresponds to the erasure pattern above is

$$\mathcal{M}_b^{(j,k,l)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & \alpha^{-j-k} & \alpha^{-j-k-l} \\ 1 & \alpha^{2j} & 0 & 0 \\ 1 & \alpha^{j} & \alpha^{j+k} & \alpha^{j+k+l} \end{bmatrix}.$$

Evaluating the determinant of this matrix gives

$$\begin{aligned} \left|\mathcal{M}_b^{(j,k,l)}\right| &= \alpha^{2j+l} + \alpha^{2j-l} + \alpha^{j-k} + \alpha^{j-k-l} + \\ &\quad + \alpha^{l} + \alpha^{-l} + \alpha^{-k} + \alpha^{-k-l} \\ &= (\alpha^{j}+1)(\alpha^{l}+1)(\alpha^{j} + \alpha^{j-l} + 1 + \alpha^{-l} + \alpha^{-k-l}) \end{aligned}$$

Here too, if $p > 5$ the product contains only non-zero terms, hence all erasure combinations of that form are correctable. □

The next Lemma treats additional erasure combinations that include parity columns and that are not covered by Lemma 2.

Lemma 4 *The following 4-erasure combinations are correctable:*

1) R_1', 1 *odd information column and* 2 *even information columns*
2) R_0, 1 *even information column and* 2 *odd information columns*
3) R_0,R_1', 1 *even information column and* 1 *odd information column, except pairs of information columns numbered* $2i, 2i+1$, *respectively, for* $1 \leq i \leq p$.

Proof: The sub-matrix that corresponds to case 1 is, up to a multiplicative non-zero constant,

$$\mathcal{M}_1^{(j,k)} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & \alpha^{-j} & 0 & 1 \\ 1 & 0 & \alpha^{2(j+k)} & 0 \\ 1 & \alpha^{j} & \alpha^{j+k} & 0 \end{bmatrix}.$$

The variables j,k satisfy the following conditions: $1 \leq k$, $1 \leq j+k \leq p-1$. Evaluating the determinant of this matrix gives

$$\left|\mathcal{M}_1^{(j,k)}\right| = \alpha^{j}(\alpha^{j+k}+1)(\alpha^{j+k}+\alpha^{k}+1),$$

an invertible element if $p > 3$.

The sub-matrix that corresponds to case 2 is, up to a multiplicative non-zero constant,

$$\mathcal{M}_2^{(j,k)} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & \alpha^{-j} & \alpha^{-j-k} & 0 \\ 1 & 0 & 0 & 1 \\ 1 & \alpha^{j} & \alpha^{j+k} & 0 \end{bmatrix}.$$

The variables j,k satisfy the following conditions: $1 \leq k$, $1 \leq j+k \leq p-1$. The determinant now equals

$$\left|\mathcal{M}_2^{(j,k)}\right| = \alpha^{-j-k}(\alpha^{k}+1)(\alpha^{j+k}+\alpha^{j}+1),$$

an invertible element if $p > 3$.

The sub-matrix that corresponds to case 3 is, up to a multiplicative non-zero constant,

$$\mathcal{M}_3^{(j)} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & \alpha^{-j} & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & \alpha^{j} & 0 & 0 \end{bmatrix},$$

whose determinant equals

$$\left|\mathcal{M}_3^{(j)}\right| = \alpha^{j}+1,$$

an invertible element if $p > 3$ and if $j > 0$. The latter condition is equivalent to requiring that the even and the odd information columns are not numbered $2i, 2i+1$, respectively, for $1 \leq i \leq p$. □

Finally, we are ready to prove the main result of this subsection. RC codes are next shown to correct all 4-erasures in up to two clusters, and asymptotically all 4-erasures in three clusters. Given the Lemmas above, establishing these results is rather straightforward.

Theorem 5 *RC codes correct all 4-erasures that fall into at most two clusters.*

Proof: If a 4-erasure falls into two or less clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). If P or Q is erased, then the remaining 3 columns cannot be all odd or all even. Lemmas 2, 3 and 4 address all such combinations, except the two special cases $\{R_1', 2, 3, R_0\}$ and $\{R_1', 2p, 2p+1, R_0\}$.

These combinations can be addressed by internal reordering of even information columns in a way that does not affect any of the other proved properties of the code (we preferred not to present the code in this reordered form, since the code structure would have been obstructed). □

Theorem 6 *For $p > 5$, the ratio between the number of RC-correctable 4-erasures that fall into three clusters and the total number of 4-erasures with three clusters is greater than 0.9696. As p goes to infinity, this ratio tends to 1.*

Proof: A 4-erasure with three clusters has two clusters of size 1 and one cluster of size 2. If a 4-erasure falls into three clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). Lemmas 2, 3 and 4 address all such combinations, except the following special cases. $\{R'_1, 2i, 2i+1, R_0\}$ cannot be corrected as it is not covered by case 3 of Lemma 4. Also, $\{P, R'_1, 2i+1, 2j+1\}$ and $\{2i, 2j, R_0, Q\}$ cannot be corrected as they are excluded from Lemma 2.

Hence the number of non-correctable 4-erasures with three clusters is

$$p + 2\binom{p}{2}$$

The total number of 4-erasures with three clusters is

$$3\binom{2p-1}{3}$$

(in general this equals $3\binom{n-3}{3}$ for length n arrays since by taking any choice of 3 points on a length $n-3$ line, we can uniquely obtain an erasure combination with three clusters, following the procedure below. We first choose 3 points from the $n-3$ line to be the cluster locations. Then the point that represents the cluster with size 2 is selected from these 3 points (for that we have the factor 3). Given these choices, the 3 clusters are obtained by augmenting the size 2 cluster with an additional point to its right and in addition augmenting each of the two left points with a point to its right as a cluster spacer.)

Thus the ratio between the number of correctable such 4-erasures and the total number of such 4-erasures equals

$$\begin{aligned}\frac{3\binom{2p-1}{3} - p - 2\binom{p}{2}}{3\binom{2p-1}{3}} &= 1 - \frac{p^2}{4p^3 - 12p^2 + 11p - 3} \\ &= 1 - \frac{9}{8p-12} - \frac{1}{8p-4} + \frac{1}{p-1}.\end{aligned}$$

For $p = 11$, the ratio attains its minimal value of 0.9696. Moreover, it is readily seen that this ratio equals $1 - o(1)$, while $o(1)$ are terms that tend to zero as p goes to infinity. □

5.2 Random erasure correctability

RC codes are next shown to correct a $7/8$ portion of all combinations of 4 erasures.

Theorem 7 *For $p > 5$, the ratio between the number of RC-correctable 4-erasures and the total number of 4-erasures is greater than 0.865. As p goes to infinity, this ratio tends to $7/8 = 0.875$.*

Proof: Building on Lemmas 2, 3 and 4, the number of correctable 4-erasures equals

$$\overbrace{2\binom{p+3}{3}(p+1) - (p+1)^2}^{(1)} + \overbrace{\binom{p}{2}\binom{p}{2}}^{(2)} + \underbrace{2p\binom{p}{2}}_{(3)} + \underbrace{p(p-1)}_{(4)}$$

(1), obtained by Lemma 2, is the number of ways to select 3 even information columns (or R_0 or P or Q) and 1 odd information column (or R'_1), multiplied by 2 to include the complement case, and subtracting the doubly counted combinations with both P and Q.

(2), obtained by Lemma 3, is the number of ways to select 2 even and 2 odd information columns.

(3), obtained by case 1 and 2 of Lemma 4, is the number of ways to select 2 even information columns and 1 odd information column, multiplied by 2 to include the complement case.

(4), obtained by case 3 of Lemma 4, is the number of ways to select an even information column $2i$ and an odd information column which is not $2i+1$.

The total number of 4-erasure combinations is

$$\binom{2p+4}{4}$$

Taking the ratio of the two we obtain

$$\frac{7p^4 + 34p^3 + 59p^2 + 32p + 12}{8p^4 + 40p^3 + 70p^2 + 50p + 12}$$

For $p = 11$, the ratio attains its minimal value of 0.865. Moreover, it is readily seen that this ratio equals $7/8 - o(1)$, while $o(1)$ are terms that tend to zero as p goes to infinity. □

6 Reliability Analysis of RC-code Protected Disk Arrays

The main motivation for the construction of array codes in general, and RC codes in particular, is to provide efficient protection for storage arrays against device failures. The benefit of deploying an array code in a practical storage system obviously lies in the trade-off it achieves between erasure correction capability and implementation complexity. To this end, the correction capability of RC codes was characterized in detail in the previous section. The advantage of RC codes in terms of implementation complexity will be established in the next two sections: section 7 and section 8. These alone may not necessarily satisfy a storage consumer, who is interested in the data *reliability* achieved by RC codes. For that purpose, we now instill the previously discussed correction capability of RC codes into a statistical framework for analyzing the reliability of the stored data. For a time instance t, the reliability of a disk array is defined as the

probability that no data has been lost at time t [8]. Finding the full reliability distribution for all times t is hard except for very simple protection structures. Therefore, the *expected* time before data loss, denoted $MTTDL$ (Mean Time To Data Loss), is used in practice as a quantitative indicator for the system reliability. Ultimately, this section will detail a procedure to find the $MTTDL$ of RC-code protected disk arrays in the presence of random and clustered device failures. This will be done after first presenting the general method of $MTTDL$ calculation as applied in the literature to MDS codes under random failures.

6.1 $MTTDL$ calculation for MDS codes under random failures

Using the method presented in [8, Ch.5] for single-erasure-correcting arrays under random failures (termed *Independent disk lifetimes* therein), we calculate the $MTTDL$ of all-4-erasure-correcting arrays as an example that will be later used for comparison with RC codes. The direct calculation of the $MTTDL$ becomes a simpler task if disk failures and repairs follow a Markov process and can thus be described by Markov state diagram. To allow that, the following assumptions are made.

- Disk lifetimes follow an exponential distribution with equal mean[2] $MTTF_{\text{disk}} = 1/\lambda$.
- Repair times are also exponential with mean $MTTR_{\text{disk}} = 1/\mu$.
- The number of disks is large compared to the number of tolerable failures so the transition probabilities between states do not depend on the instantaneous number of failed disks.

When those assumptions are met, the reliability of a disk array can be described by the state diagram shown in Figure 5. The label of each state represents the number of failed disks.

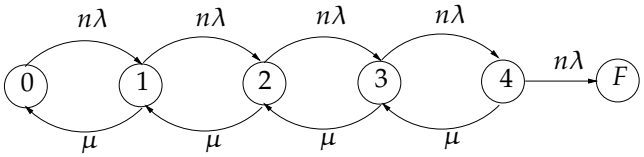

Fig. 5. State diagram description of all-4-erasure correcting arrays under random failures. The failure process with rate $n\lambda$ moves to a higher failure state. The repair process with rate μ moves to a lower failure state.

State F (Fail) represents permanent data loss resulting from a failure count that is above the array correction capability. The exponential distributions allow specifying the transitions between states in terms of *rates*. The transition rate from lower to higher states is the inverse $MTTF_{\text{disk}}$ of individual disks, times the number of disks in the array. The reverse transitions that represent repairs have rates that are the inverse $MTTR_{\text{disk}}$ assumed in the system. Using the state diagram, the $MTTDL$ is the expected time beginning in state 0 and ending on the transition into state F.

$$MTTDL \triangleq E[0 \to F]$$

2. MTTF stands for Mean Time To Failure while MTTR stands for Mean Time To Repair

The Markov property of the process permits the decomposition

$$E[0 \to F] = E[\text{time stays in } 0] + E[1 \to F] = \frac{1}{n\lambda} + E[1 \to F]$$

Linear relationships between $E[i \to F]$ and $E[j \to F]$ are induced whenever state i and state j are connected. The $MTTDL$ is then obtained as the solution (for $E[0 \to F]$) of the following linear system.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} & 0 & 0 \\ 0 & -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} & 0 \\ 0 & 0 & -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} \\ 0 & 0 & 0 & -\frac{\mu}{\mu+n\lambda} & 1 \end{bmatrix} \begin{bmatrix} E[0\to F] \\ E[1\to F] \\ E[2\to F] \\ E[3\to F] \\ E[4\to F] \end{bmatrix} = \begin{bmatrix} \frac{1}{n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \end{bmatrix}$$

that is found to be

$$MTTDL_{\text{MDS4}} = \frac{1}{\Lambda^5}(5\Lambda^4 + 4\mu\Lambda^3 + 3\mu^2\Lambda^2 + 2\mu^3\Lambda + \mu^4)$$

where $\Lambda \triangleq n\lambda$ was used for notational convenience.

6.2 $MTTDL$ calculation for RC codes under random and clustered failures

For the model of random failures, the $MTTDL$ of RC codes can be calculated by a straightforward application of the method in the previous sub-section – executed on the transition diagram of Figure 6.

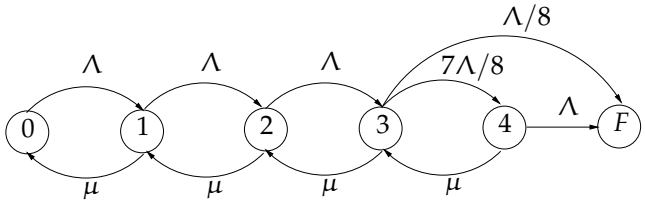

Fig. 6. State diagram description of RC-coded arrays under random failures. Since an RC code corrects only a $7/8$ ratio of 4-erasures, the failure rate out of state 3 is split to two rates with different terminal states.

The corresponding linear system of equations on the 5 active states $0, 1, 2, 3, 4$ is

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{\Lambda}{\mu+\Lambda} & 0 & 0 \\ 0 & -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{\Lambda}{\mu+\Lambda} & 0 \\ 0 & 0 & -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{7}{8}\cdot\frac{\Lambda}{\mu+\Lambda} \\ 0 & 0 & 0 & -\frac{\mu}{\mu+\Lambda} & 1 \end{bmatrix} \begin{bmatrix} E[0\to F] \\ E[1\to F] \\ E[2\to F] \\ E[3\to F] \\ E[4\to F] \end{bmatrix} = \begin{bmatrix} \frac{1}{\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \end{bmatrix}$$

The solution of that system gives

$$MTTDL_{\text{RC,rand}} = \frac{39\Lambda^4 + 35\mu\Lambda^3 + 26\mu^2\Lambda^2 + 17\mu^3\Lambda + 8\mu^4}{8\Lambda^5 + \mu\Lambda^4}$$

The exact $MTTDL$ calculations are now used to compare the reliability of RC codes to the reliabilities of all-4-erasure and all-3-erasure correcting codes. For the comparison, Λ is fixed to be $100/8760_{[1/\text{hr}]}$, which applies e.g. to an array with 100 disks and $MTTF_{\text{disk}} = 1_{[\text{Year}]}$. The $MTTDL$ in hours ([hr]) is then calculated for repair rates μ between $0.01_{[1/\text{hr}]}$ and $10_{[1/\text{hr}]}$. The graph in Figure 7 shows that RC codes outperform 3-random failure codes by an order of magnitude, despite having the same encoding complexity, the same update complexity

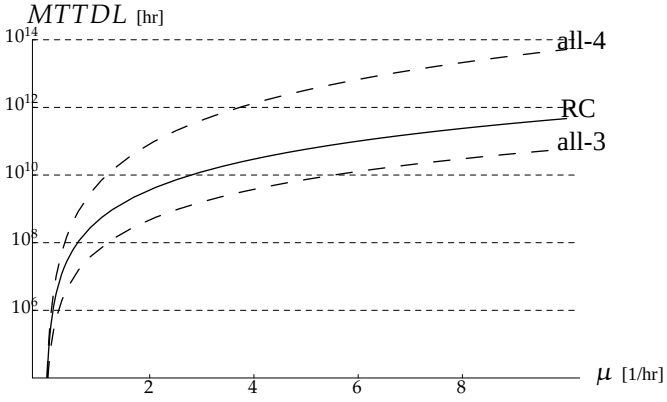

Fig. 7. $MTTDL$ curves under random failures for RC codes, all-3-erasure and all-4-erasure correcting codes. Under random failures, RC codes are order of magnitude better than all-3-erasure correcting codes, and two orders of magnitude inferior to all-4-erasure correcting codes.

and asymptotically the same decoding complexity. However, when random failures only is assumed, RC codes are still two orders of magnitude below 4-random failure-correcting codes.

To compare RC codes and 4-random failure codes in the presence of both random and clustered failures, the state diagram of RC codes in Figure 6 needs to be modified to include additional states that represent clustered failures. The state diagram of 4-random failure codes in Figure 5 remains the same since this code is oblivious to the distinction between random and clustered failures. To take clustered failures into account in the Markov failure model, we add the following assumptions to those of the previous sub-section.

- Times to clustered failures (failures that are adjacent to an unrepaired previous failure) are exponentially distributed with mean $1/\chi$.
- The exponentially-distributed repair process eliminates isolated failures before clustered ones.

With these assumptions, the state diagram of RC-code-protected arrays with random and clustered failures is given in Figure 8. States $2'$,$3'$ and $4'$ in the upper branch represent

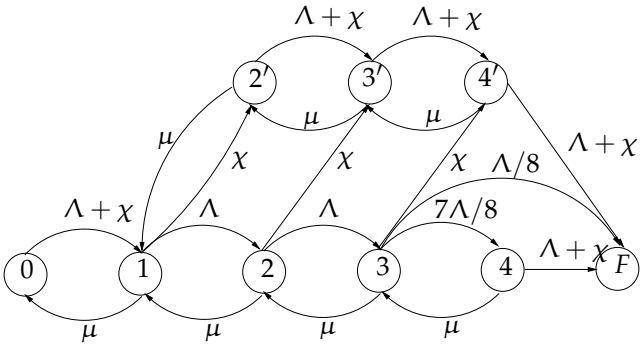

Fig. 8. State diagram description of RC-coded arrays under random and clustered failures. A new failure process with rate χ introduces clustered failures.

2, 3 and 4 clustered (not all-isolated) failures, respectively. The transitions marked with χ represent moving from all-isolated failures to a clustered failure combination. At the upper branch, both random and additional clustered failures result in clustered failure combinations – and that accounts for the transitions marked $\Lambda + \chi$. From state 0 a clustered failure is not well defined, but the rate χ is added to the transition to maintain consistency with respect to the total failure rate outgoing from all other states, which equals $\Lambda + \chi$.

Solving the 8×8 linear system for the diagram in Figure 8, the $MTTDL$ can be calculated in closed form for all combinations of χ, Λ, μ. This ability to have a closed form expression for the $MTTDL$ of RC codes, under both random and clustered failures, is crucial for a system operator to predict the reliability of the storage array under more realistic failure assumptions. The resulting $MTTDL$ curves for RC codes under three different χ values are plotted in Figure 9, and compared to the $MTTDL$ of a 4-random failure code under the same conditions (4-random codes give the same $MTTDL$ independent of the ratio between χ and Λ, as long as their sum is fixed). Not surprisingly, the curves of Figure 9 prove that as clustered failures become more dominant, the reliability of RC codes is approaching the reliability of a 4-random failure-correcting code.

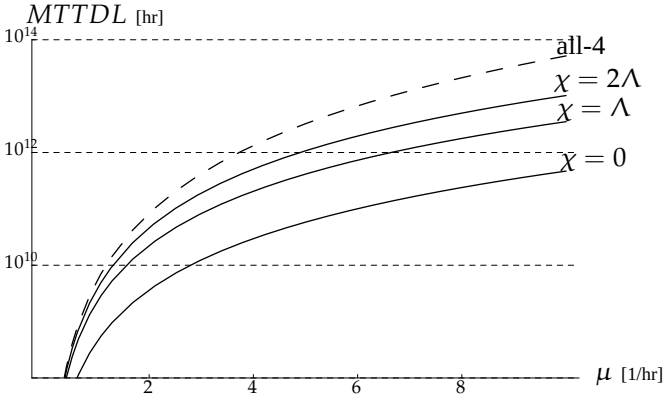

Fig. 9. $MTTDL$ curves under random and clustered failures for RC codes and all-4-erasure correcting code. For three values of χ, the $MTTDL$ of RC codes is shown by the solid curves. The $MTTDL$ of an all-4-erasure correcting code is the same for all values of χ.

7 Efficient Decoding of Erasures

In section 5, the decodability of clustered and random erasures was proved by algebraic reasoning. In this section we take a more constructive path and study simple and efficient ways to decode random and clustered erasures. The purpose of this analysis is to prove that decoding the RC code can be done using $3kp + o(p^2)$ bit operations, while the best known algorithm for a 4-erasure correcting MDS code is $4kp + o(p^2)$ [5]. Since k is taken to be in the order of p, saving about kp bit operations gives a quadratic (in p) savings in computations that is very significant in practice for large p.

For the sake of the analysis, we only consider erasure of 4 *information* columns since these are the most common and most challenging cases. We moreover only consider erasures of two even columns and two odd columns, since for RC codes, the three even and one odd case (or three odd and one even), reduces to a correction of three even (or odd)

erasures, preceded by a simple parity completion for the single odd (or even) column erasure. A very simple and efficient decoder for three odd-column erasures can be obtained by using the decoder of the STAR code, given in [10], and a same-complexity modification of that decoder can be used for the case of three even-column erasures. Throughout the section we will assume that one of the erased columns is the leftmost even information column, as all other cases are cyclically equivalent.

7.1 Description of 4-erasure decoding algorithm

A general 4-erasure can be decoded using a straightforward procedure over $\mathcal{R}_p$. Ways to perform the steps of that procedure in an efficient way are the topic of the next sub-section. The erased symbols, which represent the content of the erased array columns, are denoted by $\{e_1, o_1, e_2, o_2\}$. e_1, e_2 have even locations and o_1, o_2 have odd locations. First the syndrome vector $\boldsymbol{s}$ of the array is calculated by taking the product

$$\boldsymbol{s} = H\boldsymbol{r}$$

where $\boldsymbol{r}$ is the length $2p+4$ column vector over $\mathcal{R}_p$ that represents the array contents, with erased columns set to the zero element. Then the erased columns can be directly recovered by

$$\begin{bmatrix} e_1 \\ o_1 \\ e_2 \\ o_2 \end{bmatrix} = E^{-1}\boldsymbol{s} \tag{2}$$

where E denotes the 4×4 sub-matrix of H that corresponds to the 4 erased columns' locations:

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha_1^{-1} & 0 & \alpha_3^{-1} \\ 1 & 0 & \alpha_2^2 & 0 \\ 1 & \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix}.$$

Recall from (1) that each α_i is an element in $\mathcal{R}_p$ of the form α^{l_i}, for some $0 < l_i < p$. Therefore, E can be written as

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha^{-u} & 0 & \alpha^{-w} \\ 1 & 0 & \alpha^{2v} & 0 \\ 1 & \alpha^u & \alpha^v & \alpha^w \end{bmatrix}.$$

The inverse of E, which is used in (2) to decode erasures, is now given in a closed form

$$\begin{aligned} E^{-1} \quad = \quad & \left(\alpha^u+\alpha^v+\alpha^w+\alpha^{u+v}+\alpha^{v+w}\right)^{-1} \cdot \\ & \begin{bmatrix} 1+\alpha^v & 0 & 0 & 0 \\ 0 & \alpha^u+\alpha^w & 0 & 0 \\ 0 & 0 & 1+\alpha^v & 0 \\ 0 & 0 & 0 & \alpha^u+\alpha^w \end{bmatrix}^{-1} \cdot \\ & \begin{bmatrix} \alpha^{2v}(\alpha^u+\alpha^w) & \alpha^{u+2v+w} & \alpha^{u+\alpha^v+\alpha^w} & \alpha^{2v} \\ \alpha^{u+v} & \alpha^{u+w}(\alpha^v+\alpha^w+\alpha^{v+w}) & \alpha^u & \alpha^u(1+\alpha^v) \\ \alpha^{u+\alpha^w} & \alpha^{u+w} & 1+\alpha^{u+\alpha^w} & 1 \\ \alpha^{v+w} & \alpha^{u+w}(\alpha^u+\alpha^v+\alpha^{u+v}) & \alpha^w & \alpha^w(1+\alpha^v) \end{bmatrix} \end{aligned}$$

From (2) and the closed-form expression above, the erased symbol e_1 can be recovered by the following product

$$\begin{aligned} e_1 = & \left[\left(\alpha^u+\alpha^v+\alpha^w+\alpha^{u+v}+\alpha^{v+w}\right)\cdot\left(1+\alpha^v\right)\right]^{-1} \cdot \\ & \cdot \begin{bmatrix} \alpha^{2v}(\alpha^u+\alpha^w), & \alpha^{u+2v+w}, & \alpha^u+\alpha^v+\alpha^w, & \alpha^{2v} \end{bmatrix} \cdot \boldsymbol{s} \end{aligned}$$

Once e_1 is known, e_2 can be recovered using a simple parity completion with the aid of parity column R_0. The bits of the odd columns are then recovered by a chain of XOR operations with the aid of parity columns P, Q, that can now be adjusted to $P1, Q1$ when all even columns are known.

Calculating e_1 then reduces to the following chain of calculations

1) Finding the inverse of $\left(\alpha^u+\alpha^v+\alpha^w+\alpha^{u+v}+\alpha^{v+w}\right)\left(1+\alpha^v\right)$ over $\mathcal{R}_p$.
2) Multiplication of sparse $\mathcal{R}_p$ elements by dense $\mathcal{R}_p$ elements. The sparse elements are the four elements from the E matrix (that have a small (≤ 3) constant number of non-zero monomials, for any p) and the dense elements are the four syndrome elements.
3) Multiplication of two dense $\mathcal{R}_p$ elements resulting from the previous steps.

7.2 Analysis of 4-erasure decoding algorithm

We now analyze the number of bit operations required for each decoding task.

1) **Finding inverses of $\mathcal{R}_p$ elements**:
   The inverse of an element $f(\alpha) \in \mathcal{R}_p$ is the element $\tilde{f}(\alpha)$ that satisfies $\tilde{f}(x)f(x) + a(x)M_p(x) = 1$, for some polynomial $a(x)$. When $f(\alpha)$ is invertible, the polynomial $\tilde{f}(x)$ can be found by the Extended Euclid Algorithm for finding the greatest common divisor of the polynomials $f(x)$ and $M_p(x)$. An efficient algorithm for polynomial greatest common divisors is given in [1, Ch.8] that requires $O(p\log^4 p)$ bit operations ($O(\log p)$ polynomial multiplications, each taking $O(p\log^3 p)$ bit operations, as shown in item 3 below).
2) **Multiplication of a sparse $\mathcal{R}_p$ element by a dense $\mathcal{R}_p$ element** requires $O(p)$ bit operations. Since the number of non-zero monomials in the sparse polynomial is constant in p, the trivial polynomial multiplication algorithm requires $O(p)$ shifts and additions modulo 2.
3) **Multiplication of two dense $\mathcal{R}_p$ elements** can be done in $O(p\log^3 p)$ bit operations using Fourier domain polynomial multiplication. We describe this procedure for the special case of polynomial coefficients over GF(2). Let $N \geq 2p-2$ be the smallest such integer of the form $N = 2^\ell - 1$. Let ω be a principal Nth root of unity in the finite field GF(2^ℓ). Then ω defines a Discrete Fourier Transform on length N vectors over GF(2^ℓ). The product of two polynomials of degree $p-2$ or less can be obtained by element-wise multiplication of their individual Discrete Fourier Transforms, and then applying the Inverse Discrete Fourier Transform to the resulting length N vector. Using the FFT algorithm, each transformation requires $O(N\log N)$ operations over GF(2^ℓ), or $O(N\log^3 N)$ bit operations. The element-wise multiplication requires N multiplications over GF(2^ℓ), or $O(N\log^2 N)$ bit operations. Since $N < 4p$, the total number of bit operations needed for multiplying two dense $\mathcal{R}_p$ elements is $O(p\log^3 p)$.

8 Code Evaluation and Comparison with Existing Schemes

We compare RC codes to EVENODD ($r = 4$) codes using various performance criteria. The failure-correction properties in Table 1 apply for any prime p such that 2 is primitive in GF(p).

	RC Codes	4-EVENODD
Code Length (up to)	$2p$	p
Redundancy	4	4
Encoding Complexity	$3kp$	$4kp$
Decoding Complexity	$3kp$	$4kp$
Update Complexity	5	7
Clustered Failures	$\sim$ All	All
Random Failures	$7/8$	All

TABLE 1
Comparison of RC Codes and EVENODD Codes

The redundancy r is 4 for both codes. RC codes can support up to $2p$ information columns while EVENODD can only have up to p. Since parity columns R_0 and R'_1 each depends on half of the information columns, the encoding complexity of RC codes is $3kp$, compared to $4kp$ in EVENODD. In both cases, when k is of the same order of p, the decoding complexity is dominated by syndrome calculations (for RC codes this has been shown in section 7). Therefore, similarly to the encoding case, RC codes need about $3kp$ bit operations to decode, compared to $4kp$ for EVENODD. As for the update-complexity, RC codes are significantly more efficient. Their small-write update complexity is 5. Each of the $2p(p-1)$ updated information bits needs 3 parity updates, P, Q, R_0 for bits in even columns and P, Q, R'_1 for bits in odd columns. The $4(p-1)$ bits that belong to EO diagonals ($2(p-1)$ in Q and $p-1$ in each of R_0, R'_1) require additional $p-1$ parity-bit updates each for adjusting even/odd parities. The small-write update-complexity of RC is then obtained by averaging

$$\frac{6p(p-1)+4(p-1)^2}{2p(p-1)} = 5 - o(1)$$

Recall that EVENODD has small-write update-complexity of $2r - 1 - o(1) = 7 - o(1)$. The full-column update-complexity of RC is 3 while EVENODD's is 4. Thus RC offers a 28.57% improvement in the average number of small-writes and 25% improvement in the number of full-column updates. The fraction of clustered erasures correctable by RC codes is $1 - o(1)$, essentially the same as EVENODD's 1 fraction. Only in random erasure-correction capability are RC codes slightly inferior to EVENODD codes, the fraction of correctable random erasures is $7/8 - o(1)$ compared to 1 for EVENODD.

9 Conclusion

This paper pursues the first attempt to improve the performance of array codes by prioritizing the correctable failures based on their relative locations in the array. By doing that, we part from the abstraction of a fault tolerant storage-array as an MDS code, in the hope to achieve a more realistic trade-off between redundancy and performance. The key idea in the construction of the family of RC codes is to find a "good" "cooperating interleaving" of two codes. By "cooperating interleaving" we mean that some of the code parity bits are computed from only one constituent code, but other parity bits are computed from both codes. By "good" we mean that all 4-erasure combinations, except those that fall exclusively on one constituent code, will be correctable by the code. For the particular case addressed by RC codes, the challenge was to simultaneously correct both combinations of (3 even/odd,1 odd/even) column erasures *and* combinations of (2 even,2 odd) column erasures. Both are needed to cover all cases of clustered failures. In that respect, Pyramid codes [9] use "cooperating interleaving" in their construction. Nevertheless, these interleavings are not "good" in the sense that there are many more uncorrectable erasures beyond what allowed by the definition of "good" above.

Appendix A
Array Codes: Introductory Example

The idea behind array codes is that the code is defined on two-dimensional arrays of bits, and encoding and decoding operations are performed over the binary alphabet, using simple Exclusive OR operations. An example of an array code with two parity columns that can recover from any two column erasures is given below. The $+$ signs represent binary Exclusive OR operations. The three left columns contain pure information and the two right columns contain parity bits that are computed from the information bits as specified in the chart below.

a	b	c	$a+b+c$	$a+f+e+c$
d	e	f	$d+e+f$	$d+b+e+c$

Like encoding, decoding is also performed using simple Exclusive OR operations. For example, recovering the bits a, b, d, e at the two leftmost columns is done by the following chain of computations.

$$\begin{aligned} e &= c + (a+b+c) + (d+e+f) + (a+f+e+c) \\ &\quad + (d+b+e+c) \\ d &= e + f + (d+e+f) \\ a &= c + f + e + (a+f+e+c) \\ b &= c + a + (a+b+c) \end{aligned}$$

It is left as an exercise to verify that any two column erasures can be recovered by the code above. The small-write update complexity (the qualifier *small-write* is often omitted) of an array code is the number of parity-bit updates required for a single information-bit update, averaged over all the array information bits. In the sample code above, each of the bits a, b, d, f requires 2 parity-bit updates, and each of e, c requires 3 parity-bit updates. The update complexity of that sample code is hence $(4 \cdot 2 + 2 \cdot 3)/6 = 2.333$.

Appendix B
EVENODD Codes

B.1 Geometric description of EVENODD

The simplest way to define the EVENODD code is through its encoding rules. Given a $(p-1) \times p$ information bit array,

parity column P and parity column Q are filled using the rules shown in Figure 10, for the case $p = 5$. An imaginary row, shown unshaded, is added to the array for the sake of presenting the structure of the encoding function. Parity column P is simply the bit-wise *even* parity of the information columns (4 parity groups are marked using the 4 different shapes in Figure 10a). Parity column Q is the slope-1 diagonal parities of the information bits (whose groups are marked by shapes in Figure 10b). Whether the parity of these diagonal groups is set to be even or odd is decided by the parity of the information bits that lie on the diagonal that was left blank in Figure 10b.

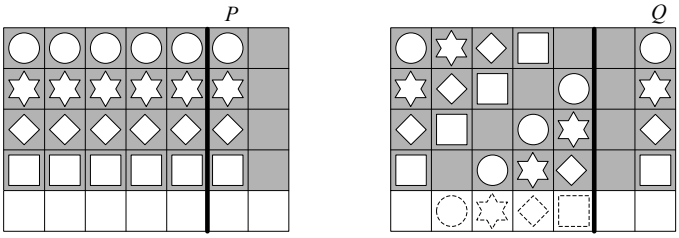

Fig. 10. Encoding of the EVENODD code. Each array of shapes specifies the encoding rules for one parity column. Each parity bit is calculated from all the information bits that carry the same shape. (a) Horizontal parity P (b) Diagonal parity Q

B.2 Algebraic description of EVENODD

In the previous sub-section, EVENODD codes were defined using their simple encoding functions. We now include the algebraic description of EVENODD codes from [2]. Columns of the $(p-1)\times(p+2)$ array are viewed as polynomials of degree $\leq p-2$ over $\mathbb{F}_2$ modulo the polynomial $M_p(x)$, where $M_p(x) = (x^p+1)/(x+1) = x^{p-1}+x^{p-2}+\cdots+x+1$ (recall that in $\mathbb{F}_2$ summation and subtraction are the same and both done using the boolean XOR function). According to that view, the polynomial for a column vector $\boldsymbol{c} = [c_0,\ldots,c_{p-2}]^T$ is denoted $c(\alpha) = c_{p-2}\alpha^{p-2}+\cdots+c_1\alpha+c_0$. Bit-wise addition modulo 2 of two columns is equivalent to summing the corresponding polynomials in the ring of polynomials modulo $M_p(x)$, denoted $\mathcal{R}_p$. Multiplying $c(\alpha)$ by α results in a downward shift of $\boldsymbol{c}$ if c_{p-2} is zero. In the case $c_{p-2}=1$, reduction modulo $M_p(x)$ is needed and $\alpha c(\alpha)$ is obtained by first downward shifting $[c_0,\ldots,c_{p-3},0]^T$ and then inverting all the bits of the shifted vector. Then, it is not hard to see that the encoding rules depicted in Figure 10 induce the following parity check matrix over $\mathcal{R}_p$.

$$H = \left[\begin{array}{cccc|cc} 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha & \cdots & \alpha^{p-1} & 0 & 1 \end{array}\right]$$

The top row of H represents the horizontal parity constraints of P, where all columns have the same alignment. The bottom row represents the diagonal parity constraints of Q, where a column is shifted one location upwards relative to its left neighbor. The structure of the ring $\mathcal{R}_p$ provides for the even/odd adjustment of the Q parities, as a function of the parity of the bits in the blank cells. The importance of this algebraic definition of the code is due to the fact that proving correctability of an erasure combination reduces to showing that the determinant of a sub-matrix of H has an inverse in the ring $\mathcal{R}_p$.

Yuval Cassuto Yuval Cassuto (S'02-M'08) is a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Laboratory. His research focuses on information theory, error-correcting codes, storage architecture and security.

He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion, Israel Institute of Technology, in 2001, and the MS and Ph.D degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively.

From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling and analysis of physical layer communication principles.

Dr. Cassuto was awarded the 2001 Texas Instruments DSP and Analog Challenge $100,000 award, as well as the Powell and Atwood graduate research fellowship awards.

Jehoshua Bruck Jehoshua (Shuki) Bruck is the Gordon and Betty Moore Professor of Computation and Neural Systems and Electrical Engineering at the California Institute of Technology. His research focuses on information theory and systems and the theory biological networks.

He received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion, Israel Institute of Technology, in 1982 and 1985, respectively and the Ph.D. degree in Electrical Engineering from Stanford University in 1989.

He has an extensive industrial experience, including working with IBM Research where he participated in the design and implementation of the first IBM parallel computer. He was a co-founder and chairman of Rainfinity, a spin-off company from Caltech that focused on software products for management of network information storage systems.

He is an IEEE fellow, and his awards include the National Science Foundation Young Investigator award and the Sloan fellowship. He published more than 200 journal and conference papers in his areas of interests and he holds more than 30 US patents. His publications were recognized by awards, including, a selection as an ISI highly cited researcher, winning the 2005 S. A. Schelkunoff Transactions prize paper award from the IEEE Antennas and Propagation society and the Best Paper Award in the 2003 Design Automation Conference.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *The design and analysis of computer algorithms*. Reading, MA USA: Addison-Wesley, 1974.

[2] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

[3] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 529–542, 1996.

[4] M. Blaum, P. Farrell, and H. van Tilborg, "Array codes," *Handbook of Coding Theory, V.S. Pless and W.C. Huffman*, pp. 1855–1909, 1998.

[5] M. Blaum and R. Roth, "New array codes for multiple phased burst correction," *IEEE Transactions on Information Theory*, vol. 39, no. 1, pp. 66–77, 1993.

[6] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *In Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, San-Francisco CA, 2004.

[7] E. Dobisz, Z. Bandic, T. Wu, and T. Albrecht, "Patterned media: nanofabrication challenges of future disk drives," *Proceedings of the IEEE: Advances in Magnetic Data Storage Technologies*, vol. 96, no. 11, pp. 1836–1846, 2008.

[8] G. Gibson, *Redundant Disk Arrays*. Cambridge MA, USA: MIT Press, 1992.

[9] C. Huang, M. Chen, and J. Li, "Pyramid codes: flexible schemes to trade space for access efficiency in reliable data storage systems," in *In Proceedings of the Sixth IEEE International Symposium on Network Computing and Applications*, Cambridge, MA USA, 2007.

[10] C. Huang and L. Xu, "Star: An efficient coding scheme for correcting triple storage node failures," in *In Proceedings of the 4th USENIX Conference on File and Storage Technologies*, San-Francisco CA, 2005.

[11] M. Kryder, E. Gage, T. McDaniel, W. Challener, R. Rottmayer, J. Ganping, H. Yiao-Tee, and M. Erden, "Heat assisted magnetic recording," *Proceedings of the IEEE: Advances in Magnetic Data Storage Technologies*, vol. 96, no. 11, pp. 1810–1835, 2008.

[12] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*. Cambridge UK: Cambridge University Press, 1986.

[13] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[14] D. A. Patterson, G. A. Gibson, and R. Katz, "A case for redundant arrays of inexpensive disks," in *Proc. SIGMOD Int. Conf. Data Management*, 1988, pp. 109–116.